- Draft -

Interactive 3D Visualization of Vector Data in GIS

Oliver Kersting
University of Potsdam
Hasso-Plattner-Institute
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
oliver.kersting@hpi.uni-potsdam.de

Jürgen Döllner
University of Potsdam
Hasso-Plattner-Institute
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
juergen.doellner@hpi.uni-potsdam.de

ABSTRACT

Vector data represents one major category of data managed by GIS. This paper presents a new technique for vector-data display that is able to precisely and efficiently map vector data on 3D objects such as digital terrain models. The technique allows the system to adapt the visual mapping to the context and user needs and enables users to interactively modify vector data through the visual representation. It represents a basic mechanism for GIS interface technology and facilitates the development of visual analysis and exploration tools.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces - *graphical user interfaces, screen design, interaction styles*. I.3.3 [**Computer Graphics**]: Picture/Image Generation - *display algorithms, viewing algorithms.* I.3.6 [**Computer Graphics**]: Methodology and Techniques - *interaction techniques.* I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism - *color, shading, shadowing, and texture.*

General Terms

Algorithms, Performance, Design.

Keywords

Vector Data, 3D GIS, Geographic Visualization, Animated Cartography.

1. INTRODUCTION

Vector data represents one of the main categories of geo-data managed by geo-information systems (GIS). Main primitives include points (e.g., cities, monuments), lines (e.g., road networks, rivers, coastlines), and polygons (e.g., national borders, vegetation zones). In the following we understand *vector data* as any 2D or 3D analytically described geo data as opposed to raster data.

There are two principal methods to visualize vector data by 2D graphics. (1) Vector data is mapped by 2D primitives such as points, lines, and polygons, which can be modified by varying point symbols, line patterns, or polygon fill-styles. (2) A set of vector data is rasterized at a given resolution as a 2D image and combined with other images (e.g., road system combined with topographic map) by 2D image operations.

To display vector data in 3D, however, these methods have several drawbacks. Most 3D terrain representations are based on a level-of-detail terrain model (e.g., [10][12]), which is needed to handle large terrain data sets (Figure 1a), and whose geometry is refined according to camera position and screen size.

We can map vector data to 3D geometric objects and integrate them in the 3D scenery. In this case, rendering artifacts are likely to occur unless vector-data is mapped consistently and exactly to the current level-of-detail. Rendering coplanar geometry, however, causes z-buffer artifacts (Figure 1b).

Another strategy is to rasterize vector data as 2D images in a pre-processing step. The image is used as 2D texture and projected onto the level-of-detail terrain geometry. Using multi-texturing, different rasterized vector data sets can be visually combined [4]. Texturing as a pixel-precise rendering technique does not produce the aforementioned artifacts in the image (Figure 1c). However, the pre-processing is time-consuming, the intermediate images require additional storage space, and the resolution cannot be changed without mapping the vector data again.

In our approach, the visual mapping of vector data is specified by scene graphs [14]. They specify the visual representation of vector data at a high level of abstraction and in a hierarchical way. The scene graphs are traversed on-demand to synthesize actual 2D images stored at different resolutions as part of a texture pyramid. Elements of the texture pyramid are used to project the visual representation of vector data on any type of 3D surface. The texture generation can take place for each frame, allowing us, therefore, to map dynamic, time-dependent vector data as well as to configure the representation according to the viewing conditions.

The strengths of our texture-based rendering of vector data include the complete decoupling of level-of-detail reference geometry and vector-data representation, an independent level-of-detail management for representations of vector-data, a high image quality due to the pixel-precise application of textures, and finally its straightforward adaptation to dynamic vector data and its support for interactive manipulation.

The paper is structured as follows: Section 2 discusses related work. Section 3 explains the displaying process. Section 4 discusses the dynamic display of vector data. Section 5 explains techniques for interactively manipulating vector-data. Section 6 gives conclusions.

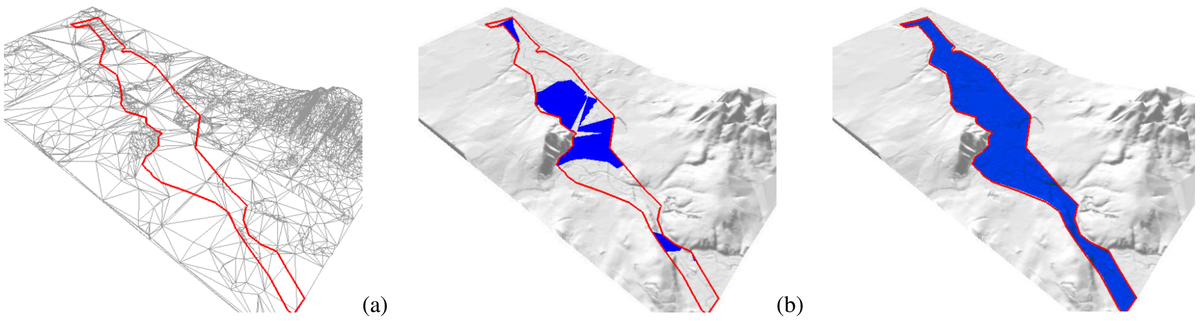
(a) (b) (c)

Figure 1. Vector data projected as polygons onto a level-of-detail terrain model, rendered in wire-frame style (a). Rendered with filled polygons: z-buffer artifacts are introduced since vector-data polygons tend to be coplanar with terrain polygons (b). Texture-based vector-data mapping does not produce artifacts (c).

2. RELATED WORK

For interactive analysis and exploration of geo data, various applications as well as extensions to GIS have been developed, e.g., ESRI´s ArcView 3D Analyst. In general, they represent vector data by 2D and 3D geometric objects superimposed on terrain models.

Frequently, maps [13] serve as tools used to communicate spatial information between GIS and users. The map metaphor has been extended to 3D [8][5]; main requirements for these 3D maps include multiresolution and multi-view representations, real-time rendering, interactivity, and high visual quality [15]. Algorithms and data structures for efficient terrain display have been extensively studied in the past (e.g., multiresolution geometry modeling [10][12] and multiresolution texture modeling [2][4]). These approaches do not offer dedicated techniques for vector-data display: Commonly, vector data is rasterized in a pre-processing step and displayed by terrain textures.

Recently, approaches towards multiresolution modeling of vector data have been emerged [1] and applied, for example, to progressively transmit or compress vector data. They do not concentrate on the display of vector data, but can substantially support the design and implementation of visual multiresolution representations of vector data.

In the field of vector-data display, the Scalable Vector Graphics language (SVG) [6], a web standard for vector-based 2D graphics, describes 2D graphics based on XML. It defines a wide variety of 2D graphics objects and styles, and concentrates on high-quality, device-independent, scalable output. SVG primarily represents a description standard but not a rendering technique. SVG does not generalize to 3D graphics, which is our main focus. SVG documents, of course, can be translated into a representation suitable for our mapping technique.

In real-time 3D computer graphics, texturing emerges as a fundamental graphics operation; applications include projective textures [7], shadows and reflections [11], multi-texturing [17], near-realistic lighting and shading [9]. Wynn [18] describes off-screen 3D rendering implemented by the OpenGL P-buffer. The P-buffer enables applications to use the full range of OpenGL for synthesizing images in an internal, non-visible framebuffer. It represents the technical basis of our approach for dynamically generating vector-data textures. In general, texture-based rendering techniques improve drastically visual quality, exactness, and expressiveness of real-time renderings.

3. DISPLAYING VECTOR DATA IN 3D

Mapping vector data to 3D requires 3D surfaces having the role of a *geo-reference surface*. In the following, we assume that a multiresolution terrain model is used as geo-reference surface.

3.1 Geometry-Based Mapping

If we represent vector data by geometric objects (e.g., 2D line segments), these objects must be linked to the geo-reference surface. In general, multiresolution modeling is necessary for representing geo-reference surfaces in order to reduce their geometric complexity and to achieve real-time rendering. However, for the vector-data mapping it is difficult (sometimes impossible) to get access to the current state of the geo-reference

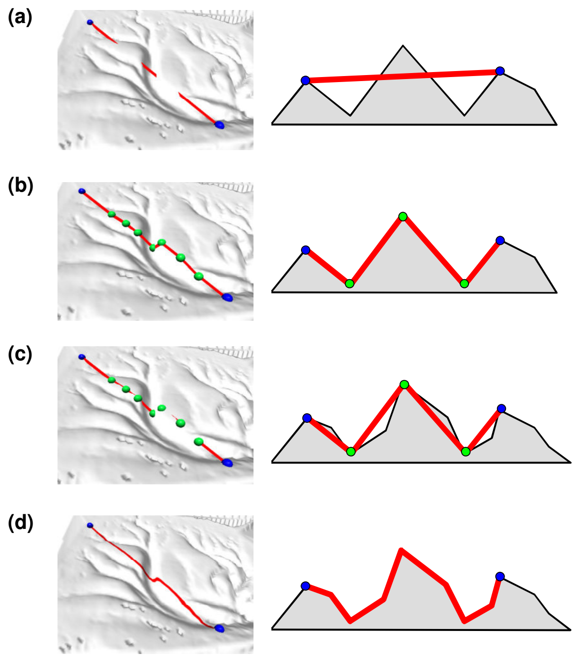

Figure 2. Geometry-based mapping of vector data and its pitfalls (a-c). Texture-based mapping of vector data (d).

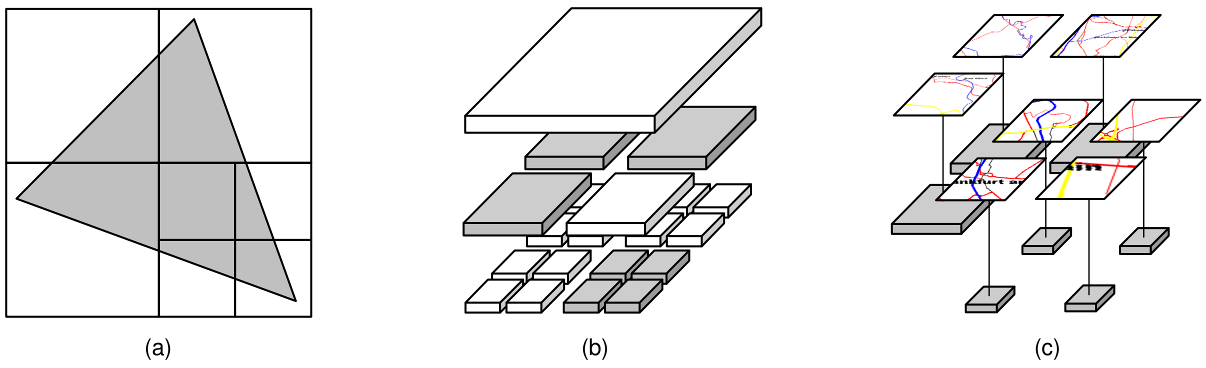

Figure 3. Texture-based mapping of vector data. (a) Top-view. The gray area indicates the current view-frustum seen by the camera. (b) Quad-tree-based decomposition of the terrain geometry. Patches near the camera have higher resolution than patches far away from the camera. The currently used patches are colored gray. (c) Corresponding collection of applied textures, derived from the same texture pyramid.

surface and to install callbacks that could transform and adapt geometric objects representing vector data so that they correspond to that state. Without this kind of callbacks changes in the level-of-detail terrain model would lead to visual artifacts.

Figure 2 illustrates the limitations of geometry-based mapping. A pipeline, defined by two geo-referenced end-points, should be visualized on top of a digital terrain model. The pipeline is represented by a line segment, which interferes with the level-of-detail terrain model (Figure 2a). We can crack pipeline segments to ensure that the visual pipeline representation tightly follows the terrain surface (Figure 2b), but the surface varies depending on viewer position and screen resolution (Figure 2c). If we would like to adapt pipeline segments to the surface, non-trivial analytic calculations would become necessary.

3.2 Texture-Based Mapping

To overcome the limitations of a geometry-based representation, we represent vector data by 2D textures that are projected onto the reference geometry. The textures result from rendering a scene graph [14] that describes the visual mapping of the vector data, called *vector-data scene graph*.

A vector-data scene graph consists of nodes. The nodes can contain 2D geometry objects (e.g., points, lines, polygons, curves), graphics attributes (e.g., color, material, textures, line style, facet style etc.), and child nodes. Subgraphs can be shared, i.e., a node can have more than one parent node. This way, complex scene objects, for example, glyphs and symbols, can be designed in a hierarchical and reusable way. Vector-data scene graphs mostly contain 2D graphics. In our implementation, we use the scene graph of the Virtual Rendering System (VRS) [3]. As the primary difference to a regular scene graph, the vector-data scene graph is attached to a P-buffer canvas.

In the example, we represent the pipeline by a vector-data scene graph that consists of a 2D line, attributed by a red color, and two 2D points, attributed by a blue color. It is rendered into a 2D texture that is projected onto the terrain surface (Figure 2d). The texture-based visual mapping of the pipeline is independent from the level-of-detail of the reference geometry, lines and points are drawn perspectively correct, and no rendering artifacts due to co-planarity and surface intersections occur.

3.3 Multiresolution Texturing

The texture-based approach can be optimized with respect to its visual quality in the case of a level-of-detail terrain model as reference geometry. We assume that the terrain model has a quad-tree structure. Each level of detail consists of terrain patches, each covering the area of all its four child patches (Figure 3b) and having higher resolution than its parent patch. The rendering algorithm determines visible patches according to the view frustum and selects patches according to quality criteria such as camera distance (Figure 3a).

3.3.1 Static Texture Pyramid

The idea is to visually map vector data at different levels of detail. Vector data is mapped into a possibly very large 2D texture in a pre-processing step. Then, the original 2D texture is down scaled at various resolutions, building a (static) *texture pyramid* [16].

Each patch of the multiresolution terrain surface corresponds to a subregion (called texture patch) of each 2D texture contained in the texture pyramid. To render a terrain patch, the rendering algorithm activates the associated texture patch (Figure 3c).

Displaying vector data based on a static texture pyramid has the following limitations with respect to quality, speed, and hardware resources: The pre-processing of the pyramid is generally not

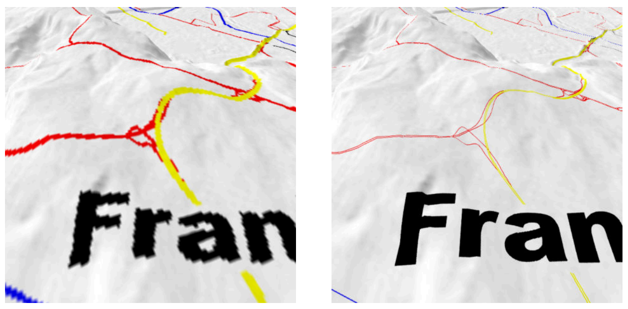

Figure 4. Differences in visual quality between static and on-demand texture pyramids. Using the static texture pyramid based on a 4096^2-sized image (left); using the on-demand texture pyramid (right).

hardware-accelerated, resolution and visual quality is fixed (Figure 4 left), the content of a selected layer cannot be changed dynamically, and memory requirements are high. The total amount of memory sums up to the original texture size plus 33%.

3.3.2 On-Demand Texture Pyramid

The *on-demand texture pyramid* has been developed to overcome the limitations of traditional texture pyramids for visual mappings of vector data. The pyramid is derived from a vector-data scene graph. If a specific texture patch is requested from the terrain rendering algorithm, the vector-data scene graph is rendered for that region. Note that for each patch, regardless of its quad-tree level, the same texture size is used, i.e., the resolution of the visual mappings of vector data increases with the level of the quad-tree.

This technique does not require pre-processing because textures are generated on the fly. Hence, memory requirements are lower because no texture pyramid must be kept in memory. The generation of a texture patch can take place for each frame, allowing us, therefore, to map dynamic, time-dependent data as well as to configure the mapping according to the viewing conditions. This way, resolution and visual quality can be adjusted to screen resolution and user needs (Figure 4 right).

3.4 Real-Time Rendering of Texture-Mapped Vector Data

The visual representation of the vector data consists of collections of graphics shapes and graphics attributes that are hierarchically arranged by scene graphs. Since scene graphs can be constructed, modified, and rendered in real-time, our approach is applicable to dynamic vector data as well and, furthermore, enables interactive manipulation of vector data (see Section 5).

The on-demand texture pyramid uses a caching mechanism to speed up the texture-patch generation process: If vector-data content of a texture patch has not changed from one frame to another, the cached texture patch is re-used.

In our implementation, the on-demand rendering of textures is based on the OpenGL pixel buffer [18]. The P-buffer is a fully functional frame-buffer, i.e., it consists of color buffer, a z-buffer, and optionally a stencil-buffer. Rendering to the P-buffer is as fast as rendering to an on-screen canvas; there is no restriction with respect to hardware acceleration. The P-buffer contents can be copied directly to a 2D texture. The P-buffer resides on graphics hardware; no texture data must be transferred to the application memory during that copy action.

Hence, copying P-buffer contents into 2D textures (also resident on graphics hardware) is extremely fast. Since P-buffer rendering allows us to rasterize vector data within real-time, on-demand generation of textures becomes practicable.

To achieve a given screen-space texture resolution, most notably, the memory requirements are drastically lower using an on-demand texture pyramid compared to a static texture pyramid. In Table 1, memory requirements of both approaches are compared. We assume that vector data is rendered with an average number of 10-25 equally sized 2D textures, which are generated on demand. In comparison, a static texture pyramid for a four level quad-tree and a source image size of 8192×8192 pixels would require

Table 1. Resource usage for different texture pyramids.

Texture Resolution	Quad-tree Levels	Min/Avg Number of Patches	Min/Avg Memory Usage MB	Resolution of Pre-Built 2D Images	Memory Usage MB
256^2	4	10/14	*2.0/2.8*	4096^2	*66*
512^2	4	10/14	*7.9/11*	8192^2	*268*
256^2	6	16/20	*3.1/3.9*	16384^2	*1073*
512^2	6	16/20	*12.6/15.7*	32768^2	*4295*
256^2	8	22/25	*4.3/4.9*	65536^2	*17180*
512^2	8	22/25	*17.3/19.7*	131072^2	*68719*

268 MB of memory to store all mip-map levels; the equally resolved on-demand texture pyramid requires 11 MB (Table 1).

4. DYNAMIC DISPLAY OF VECTOR DATA

Each texture patch of an on-demand texture pyramid can be rebuilt for each frame. Consequently, we are able to dynamically adapt visual mappings to user's needs, camera settings, or screen-space quality criteria. This enables a wide range of dynamic visualization strategies, including:

- **Enabling/Disabling of Vector Data.** Each node component of a vector-data scene graph can be enabled respectively disabled. That way, users and applications can select which vector-data elements to be mapped, or reduce/increase the visual complexity of a mapping.
- **Animated Display of Vector Data.** Vector data resulting from or controlled by geo-processes or simulations (time-dependent vector data) can be represented in an animated way.
- **View-Dependent Displaying of Vector Data.** While rasterizing vector data we can take into account the graphics context such as camera settings (distance, orientation), view-frustum culling, screen-space extension etc. to control visual appearance and design of vector data (Figure 5).
- **Mixing Texture-Based and Geometry-Based Mapping of Vector Data.** There are two basic strategies for selecting 2D or 3D representations: We can opt for a mixed representation (1) when vector-data elements come close to the viewer to show more details by 3D objects, or (2) when vector data elements move far away from the viewer to ensure that they remain visible by 3D objects. The first strategy can be

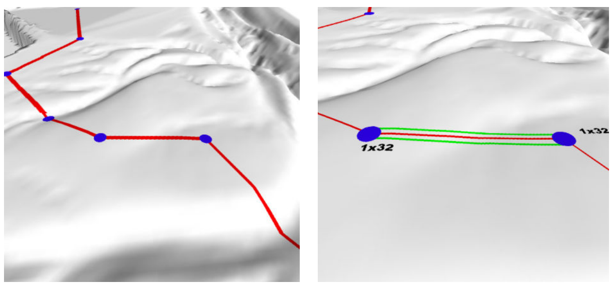

Figure 5. Visual mapping of a pipeline at low level-of-detail (left). When the camera gets closer to one pipeline segment, details, such as capacity and name, become visible (right).

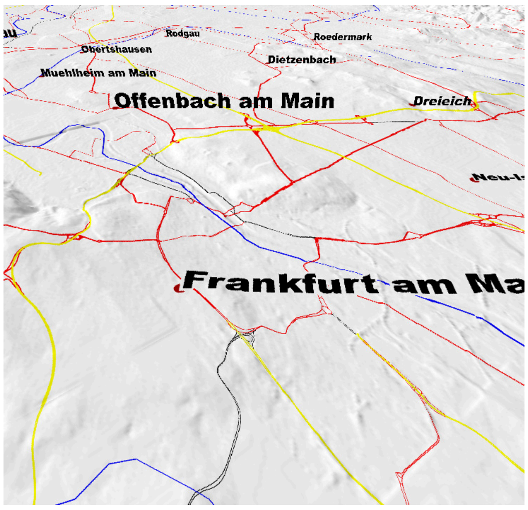

Figure 6. Hybrid representation of labels inside 2D texture and as 2D text objects. Far-away labels are displayed as billboards, they are not contained as elements of the terrain texture.

applied to complex 3D objects (e.g., building models), which can be abstracted in the 2D representation. The latter can be preferred in the case of objects that would become invisible (e.g., labels in Figure 6).

- **Cartographic Generalization.** As an immediate application of on-demand rasterization, applications can implement generalization schemes such as defined by cartography.

4.1 Display of Labels

Labels are frequently used elements in geo-visualizations. Commonly, labels are represented as part of a static terrain texture. They keep, therefore, orientation and resolution independently from the viewing situation. Labels as elements of an on-demand texture, however, should always be oriented towards the viewer and their font size must stay within a certain range to ensure readability (Figure 6). Also, labels could be represented as 2D text placed parallel to the view plane once the terrain surface is seen below a certain viewing angle at a far distance from the camera (Figure 7). This strategy ensures an optimal visibility of labels.

4.2 Display of Time-Dependent Vector Data

In the case of time-dependent vector data, we must update their corresponding elements in the vector-data scene graph. In general, it will not be necessary to rebuild the whole scene graph. Thus dynamic phenomena like flooding scenarios, air pollution processes, or traffic simulations can be modeled in a straightforward way.

5. INTERACTIVE MANIPULATION OF VECTOR DATA

Interaction with vector data is necessary to let the user manipulate that data. Since vector data is given in an analytic form (not rasterized), the semantics of the data is directly available for any kind of interactive editing, direct manipulation, and exploration tools.

5.1 Picking Operation

In common 3D libraries, 3D interaction is supported by picking operations. One kind of picking implementation consists in constructing a 3D ray that is sent through a point in the view-plane into the 3D viewing frustum, and testing the ray for intersection with scene objects.

The picking operation starts with a primary ray test: a 3D ray passes the view-plane through the perspective view-frustum. Along its way, the picking request can hit none, one, or several 3D scene objects. If scene objects are hit, the intersection point, together with the object-id is recorded. The picking operation, in general, returns a list of all objects hit together with the intersection points.

If the ray hits the terrain surface, we check for vector-data elements. For it, we send a new, redirected ray through the virtual (2D) scene described by the vector-data scene graph to check for intersections with the visual representations of vector data (Figure 8). The identifiers of hit shapes are returned and can be traced back to vector-data elements. Note that vector-data scene graphs can contained and their pick even invisible vector-data elements such as groups or regions.

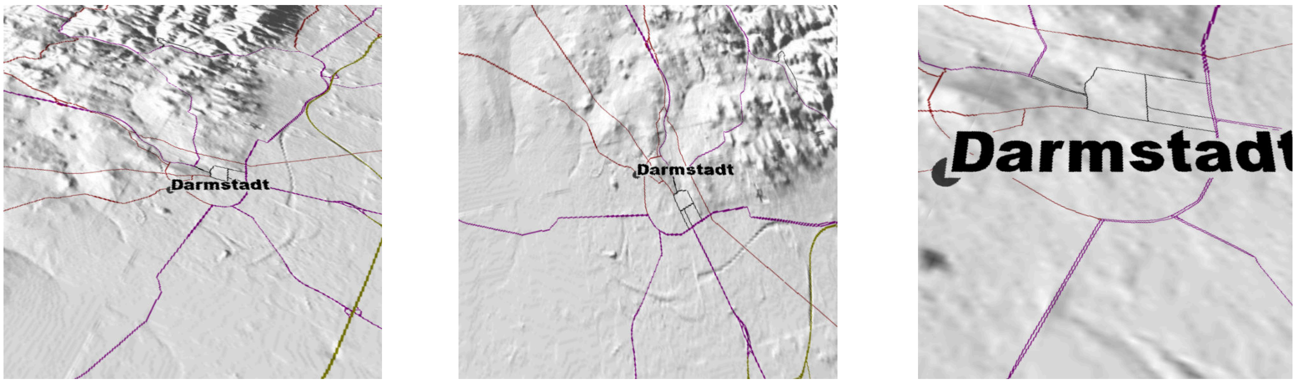

Figure 7. Texture-based, automatically oriented labels contained in a terrain texture.

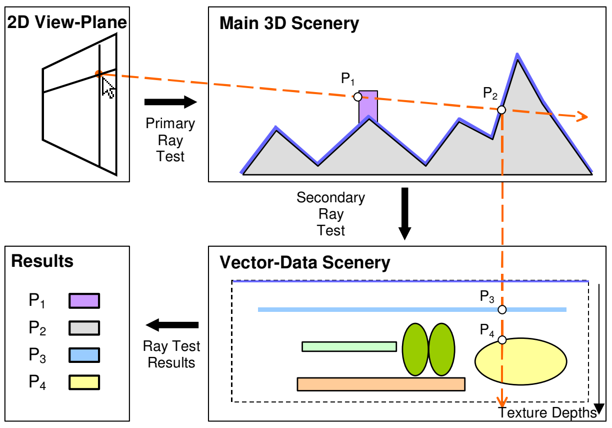

Figure 8. Picking of vector data in their visual representation: 3D-Ray tests are re-directed from the main scene graph to the vector-data scene graph.

5.2 Manipulation of Vector Data

Direct manipulation of visualized vector data represents a powerful technique to enable interaction of users with visually mapped vector data. The interaction requirement – to react in real-time to user actions – is fulfilled because of the real-time ability of on-demand texture pyramids.

2D graphics editing operations such as selecting, moving, resizing, rotating, and scaling can be implemented for geo-objects described by vector data. In general, it is necessary to add visual handles to the vector-data scene graph (or main scene graph). The user can directly manipulate the handles and, thereby, control the corresponding operation. In addition, an interaction mechanism can apply the picking operation of vector-data scene graphs to determine which objects the user selects. For example, to edit a polygon, its vertices are represented by 3D spheres, which the user can move over the terrain surface. The application is responsible for synchronizing vector data, vector-data scene graphs, and visual handles.

6. CONCLUSIONS

In our approach, we specify the visual mapping of vector data by scene graphs, render them in 2D textures, and project these textures onto geo-reference geometry such as terrain surfaces on a per-frame basis. The approach benefits from texturing as a fundamental high-quality and hardware-accelerated graphics operation as well as from the analytic specification of the visual representation of vector data using the whole capabilities of modern scene graph libraries.

The on-demand generation of visual mappings drastically reduces the required amount of memory compared to pre-rasterizing vector data, can adapt quality to viewing conditions, and enables mapping of dynamic vector data. The scene graph representation keeps the semantics of vector data accessible in 3D, which is required for any kind of interactive 3D editing and direct manipulation of vector data. As future work, we want to integrate mechanisms for coupling 2D and 3D representations more tightly and to develop a visual mapping language.

Acknowledgement

We would like to thank Konstantin Baumann for his collaboration and work in the LandExplorer project.

REFERENCES

[1] Bertolotto, M., and Egenhofer, M.J. *Progressive Vector Transmission.* Proceedings ACM GIS '99, 152-157, 1999.

[2] Cline, D., and Egbert, P. *Interactive Display of Very Large Textures.* Proceedings IEEE Visualization '98, 343-350, 1998.

[3] Döllner, J., and Hinrichs, K. *A Generic 3D Rendering System.* IEEE Transactions on Visualization and Computer Graphics, 8(2):99-118, 2002.

[4] Döllner, J., Baumann, K., and Hinrichs, K. *Texturing Techniques for Terrain Visualization.* Proceedings IEEE Visualization 2000, 227-234, 2000.

[5] Döllner, J., and Kersting, O. *Dynamic 3D Maps as Visual Interfaces for Spatio-Temporal Data.* Proceedings ACM GIS 2000, 115-120, 2000.

[6] Eisenberg, J.D. *SVG Essentials.* O'Reilly, 2002.

[7] Haeberli, P., and Segal, M. *Texture Mapping as a Fundamental Drawing Primitive.* Proceedings of the 4th Eurographics Workshop on Rendering, M. Cohen, C. Puech, F. Sillion (Eds.), 259-266, 1993.

[8] Haeberling, C. *Symbolization in Topographic 3D Maps: Conceptual Aspects for User-Oriented Design.* 19th International Cartographic Conference, 1037-1044, 1999.

[9] Heidrich, W., and Seidel, H.P. *Realistic, Hardware-accelerated Shading and Lighting.* Computer Graphics (Proc. SIGGRAPH '99), 171-178, 1999.

[10] Hoppe, H. *Smooth View-Dependent Level-of-Detail Control and Its Application to Terrain Rendering.* IEEE Visualization '98, 35-42, 1998.

[11] Kilgard, M.J. *Improving Shadows and Reflections via the Stencil Buffer.* NVIDIA White Paper, 2000.

[12] Lindstrom, P., and Pascucci, V. *Visualization of Large Terrains Made Easy.* Proceedings of IEEE Visualization 2001, 363-370, 2001.

[13] MacEachren, A.M. *How Maps Work: Representation, Visualization, and Design.* Guilford Press, New York, 1995.

[14] Sowizral, H. *Scene Graphs in the New Millennium.* IEEE Computer Graphics and Applications, 20(1):56-57, 2000.

[15] Terribilini, A. *Maps in Transition: Development of Interactive Vector-Based Topographic 3D-Maps.* 19th International Cartographic Conference, 993-1001, 1999.

[16] Williams, L. *Pyramidal Parametrics.* Proceedings of SIGGRAPH '83, 17(3):1-11, 1983.

[17] Woo, M., Neider, J., Davis, T., and Shreiner, D. *OpenGL Programming Guide - 3rd ed.* Addison-Wesley, 1999.

[18] Wynn, C. *Using P-Buffers for Off-Screen Rendering in OpenGL.* NVidia Technical Paper, 2002.